Decision Heuristics and Restarts in SAT

Armin Biere
Johannes Kepler University
Linz, Austria

Deduktionstreffen 2016
Alpen-Adria-Universität Klagenfurt, Austria
Monday, September 26, 2016

Supported by Austrian Science Fund (FWF) under NFN Grant S11408-N23 (RiSE)
Two Parts

[SAT’15]  with Andreas Fröhlich

- Evaluating CDCL Variable Scoring Schemes
  - attempt to simplify and explain decision heuristics in SAT
  - shows that VMTF is as good as VSIDS (and explains boths)

[POS’15]  with Andreas Fröhlich

- Evaluating CDCL Restart Schemes
  - simplifies the state-of-the-art Glucose style restart scheme
  - evaluation shows clear benefits but also weaknesses
[SAT’15]  with Andreas Fröhlich

- Evaluating CDCL Variable Scoring Schemes
  - attempt to simplify and explain decision heuristics in SAT
  - shows that VMTF is as good as VSIDS (and explains boths)

[POS’15]  with Andreas Fröhlich

- Evaluating CDCL Restart Schemes
  - simplifies the state-of-the-art Glucose style restart scheme
  - evaluation shows clear benefits but also weaknesses
Motivation Part I

- (E)VSIDS decision heuristic
  - (Exponential) Variable State Independent Decaying Sum (VSIDS)
  - empirically one of the most important features of state-of-the-art solvers
  - no formal argument “why it works”

- reconsider simpler alternatives
  - particularly variable move to front schemes (VMTF)
  - requires careful data structure design
  - formalization of these heuristics
  - empirical evaluation

- so a step towards “Trying to Understand the Power of VSIDS”
  - … and thus “Why CDCL Works”

Decision Heuristics

- decision heuristics consist of
  - variable selection: which variable to assign next?
  - phase selection: assign variable to which phase (true or false)?

- **phase saving** [PipatsrisawatDarwiche’07]
  - select phase to which variable was assigned before
  - initialized by static one-side heuristics [JeroslowWang’90]
  - very effective and thus default in state-of-the-art solvers

- we consider only variable selection as decision heuristic here
  - clause based heuristics less effective (BerkMin, CMTF)
  - same applies to literal based heuristics (using literal scores)

- variable selection and **decision heuristic** boils down to
  - compute and maintain heuristic scores for variables
  - select variable with highest score
Variable Scoring Schemes

- how to compute scores
  - static or dynamic
  - **bump** variables: when to **increase** scores and by how much
  - **rescore** variables: when to **decrease** scores and by how much
  - state-of-the-art: VSIDS (from Chaff)
    more precisely the exponential variant (EVSIDS) of MiniSAT!

- data structures for finding decision variables
  - eager or lazy update of “order”
  - state-of-the-art: priority queue of variables ordered by score (MiniSAT)

- data structure depends on how scores are computed and vice versa
Variable Scoring Schemes

- **Zero** order scheme = static scores
  - computed for instance once during preprocessing
  - still needs search for “best” unassigned variable
  - only total orders considered so far

- **First** order schemes = dynamic but state less
  - for instance: score = pos occs × neg occs
  - independent of how search reached current branch / search node
  - might be quite expensive to compute / update (linear in CNF size)

- **Second** order schemes: variable score depends on history of search
  - first order + learning ⇒ second order
  - but can also be used to speed up search for “best” variable
  - goal is logarithmic or even constant algorithm for variable selection
VSIDS appeared in seminal Chaff paper from Princeton (2001)

bump variables occurring in learned clauses
  - bumping means incrementing an integer VSIDS score
    - current state-of-the-art: bump all variables used to derive learned clause
  - independent of state of clauses (satisfied or not)

rescoring gives focus on recently used variables
  - scores are “decayed”, e.g., originally divided by two every 256th conflict
  - “low pass filter” on “use frequency” of variables

search for next unassigned variable with largest score
  - keeps an array of variables sorted by score
  - only re-sorts it w.r.t. score during rescoring (every 256th conflict)
  - uses right-most unassigned variable, thus original implementation imprecise
Siege SAT solver [Ryan'04] used variable move to front (VMTF)
- bumped variables moved to head of doubly linked list
- search for unassigned variable starts at head
- variable selection is an online sorting algorithm of scores
- classic “move-to-front” strategy achieves good amortized complexity

- original implementation severely restricted
  - only moved a subset of bumped variables
  - details about caching the search not described
    - no source code published either
  - not exactly the same as VSIDS

- as consequence VMTF not used in state-of-the-art solvers
MiniSAT’s Exponential VSIDS (EV$\text{SIDS}$)

- **floating point scores**
  - allows fine grained rescore at every conflict
  - consider multiplying by $f = 0.9$ every score at each conflict

- actually, instead of updating scores of all variables (at every conflict)
  - only increase score of bumped variables by $g^i$
  - with $i$ the conflict-index, and $g = 1/f$
  - non-bumped variables not touched

- priority queue of variables ordered by score
  - implemented as **binary heap** with update (bump and bubble up)
  - lazy assigned variable removal
    - remove largest score variable from heap until unassigned one found
    - put unassigned variables not on the heap back (logarithmic complexity)

- normalized VSIDS (NVSIDS) $\in [0, 1]$ as (theoretical) model [Biere’08] + video
### Summary Variable Scoring Schemes

Evaluating CDCL Variable Scoring Schemes

<table>
<thead>
<tr>
<th>variable score $s'$ after $i$ conflicts</th>
<th>bumped</th>
<th>not-bumped</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>static decision order</strong></td>
<td>$s$</td>
<td>$s$</td>
</tr>
<tr>
<td><strong>increment scores</strong></td>
<td>$s + 1$</td>
<td>$s$</td>
</tr>
<tr>
<td><strong>sum of conflict-indices</strong></td>
<td>$s + i$</td>
<td>$s$</td>
</tr>
<tr>
<td><strong>original implementation in Chaff</strong></td>
<td>$h_i^{256} \cdot s + 1$</td>
<td>$h_i^{256} \cdot s$</td>
</tr>
<tr>
<td><strong>normalized variant of VSIDS</strong></td>
<td>$f \cdot s + (1 - f)$</td>
<td>$f \cdot s$</td>
</tr>
<tr>
<td><strong>exponential MiniSAT dual of NVSIDS</strong></td>
<td>$s + g^i$</td>
<td>$s$</td>
</tr>
<tr>
<td><strong>average conflict-index decision scheme</strong></td>
<td>$(s + i)/2$</td>
<td>$s$</td>
</tr>
<tr>
<td><strong>variable move-to-front</strong></td>
<td>$i$</td>
<td>$s$</td>
</tr>
<tr>
<td><strong>variable move-to-front variant</strong></td>
<td>$b$</td>
<td>$s$</td>
</tr>
</tbody>
</table>

$$0 < f < 1 \quad g = 1/f \quad h_i^m = 0.5 \quad \text{if } m \text{ divides } i \quad h_i^m = 1 \text{ otherwise}$$

$i$ conflict index \quad $b$ bumped counter
Fast VMTF Implementation

- fast simple implementation for caching searches in VMTF
  - doubly linked list does not have positions as an ordered array
  - bump = move-to-front = dequeue then insertion at the head

- time-stamp list entries with insertion time
  - maintained invariant: all variables right of next-search are assigned
  - requires update to next-search while unassigning variables

```
idx: 5
val: x
time: 6

idx: 3
val: 1
time: 8

idx: 4
val: 0
time: 9

idx: 9
val: 1
time: 12

idx: 7
val: 0
time: 15

next-search

unassign 9

bump 4

next-search'
```

Decision Heuristics and Restarts in SAT @ Deduktionstreffen 2016
solved SAT competition 2014 application track instances (ordered by time)
solved SAT competition 2014 application track instances (ordered by time)
SA T'16 Competition Application Benchmarks (sorted by percentage run−time)

- VMTF bumping run−time percentage
- EVSIDS bumping run−time percentage
- EVSIDS deciding run−time percentage
- VMTF deciding run−time percentage
surveyed and classified variable selection / scoring schemes

- came up with a new one ACIDS (as well as SUM)
- EVSIDS, VMTF, ACIDS comparable in performance
- with a generic fast queue implementation

VMTF was considered to be obsolete

- can be made effective (with less code than EVSIDS)
- needs proper optimized implementation: time-stamping with insertion-time
- VMTF might be easier to reason about in proof complexity

threads to validity

- unclear whether VMTF only works in combination with Glucose restarts
  see also our POS’15 paper and talk in Part II of this talk
- benchmark selection in recent SAT competitions highly controversial

Splat: SAT solver only based on VMTF

http://fmv.jku.at/splat
[SAT’15]  with Andreas Fröhlich

- Evaluating CDCL Variable Scoring Schemes
  - attempt to simplify and explain decision heuristics in SAT
  - shows that VMTF is as good as VSIDS (and explains boths)

[POS’15]  with Andreas Fröhlich

- Evaluating CDCL Restart Schemes
  - simplifies the state-of-the-art Glucose style restart scheme
  - evaluation shows clear benefits but also weaknesses
Lingeling actually barely won
- only for long time limit of 5000 seconds
- for 900 seconds: no chance

Two main reasons
- selected benchmark biased towards descendants of Glucose / MiniSAT
- but Glucose restarts are important for many (selected) benchmarks

The POS’15 paper is about lessons learned while
- porting the Glucose restart scheme to Lingeling
- and simplifying by
  - using exponential moving averages (EMA)

SAT Competition 2014 application track instances clustered in **buckets** (by the organizers):

- 2d-strip-packing (4), argumentation (20), bio (11),
- crypto-aes (8), crypto-des (7), crypto-gos (9),
  - crypto-md5 (21), **crypto-sha (29)**, crypto-vpmc (4),
- diagnosis (28), fpga-routing (1),
- hardware-bmc (4), hardware-bmc-ibm (18), **hardware-cec (30)**,
  - hardware-manolios (6), hardware-velev (27),
- planning (19), scheduling (30), scheduling-pesp (3),
- software-bit-verif (9), software-bmc (6), symbolic-simulation (1), termination (5)

In total **300** instances clustered in **23** buckets
Status run_CDCL_loop_with_restarts () {
    for (;;) {
        if (bcp ()) {
            if (restarting ()) restart ();
            else if (!decide ()) return SATISFIABLE;
        } else {
            conflicts++;
            if (!analyze ()) return UNSATISFIABLE;
        }
    }
}

- run BCP and conflict analysis (including learning) until completion
- restart if restart policy implemented in restarting says so
- usually based on a global conflicts counter
- otherwise pick next decision (unless all are assigned)
bool restarting () {
    return conflicts >= limit;
}

void static_uniform_restart () {
    restarts++;
    limit = conflicts + interval;
    backtrack (0);
}

void static_geometric_restart () {
    limit = conflicts + interval * pow (1.5, ++restarts);
    backtrack (0);
}

void luby_restart () {
    limit = conflicts + interval * luby (++restarts);
    backtrack (0);
}

Decision Heuristics and Restarts in SAT @ Deduktionstreffen 2016
Restart Scheme Classification

- **static schemes**
  - fixed schedule of restarts only based on conflicts counter
    - **uniform intervals**: wait a fixed number of conflicts after each restart
  - non-uniform restart intervals
    - number of performed restarts determines next interval (in terms of conflicts)
    - arithmetically or geometrically increasing actual interval
    - Luby scheme (also known as reluctant doubling)
    - inner-outer scheme

- **dynamic schemes**
  - agility based restart blocking
  - local restarts (not discussed in the paper nor the talk)
  - reusing the trail implicitly also blocks restarts (even partially)
  - Glucose restart scheme (focus here)
### Comparing Static Uniform Restart Schemes

#### Evaluating CDCL Restart Schemes

<table>
<thead>
<tr>
<th>$r$</th>
<th>002</th>
<th>004</th>
<th>008</th>
<th>016</th>
<th>032</th>
<th>064</th>
<th>128</th>
<th>256</th>
<th>512</th>
</tr>
</thead>
<tbody>
<tr>
<td>tot</td>
<td>122</td>
<td>127</td>
<td>139</td>
<td>144</td>
<td>144</td>
<td>161</td>
<td>163</td>
<td>168</td>
<td>158</td>
</tr>
<tr>
<td>sat</td>
<td>40</td>
<td>41</td>
<td>49</td>
<td>56</td>
<td>56</td>
<td>73</td>
<td>76</td>
<td>83</td>
<td>79</td>
</tr>
<tr>
<td>uns</td>
<td>82</td>
<td>86</td>
<td><strong>90</strong></td>
<td>88</td>
<td>88</td>
<td>88</td>
<td>87</td>
<td>85</td>
<td>79</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$r$</th>
<th>002</th>
<th>004</th>
<th>008</th>
<th>016</th>
<th>032</th>
<th>064</th>
<th>128</th>
<th>256</th>
<th>512</th>
</tr>
</thead>
<tbody>
<tr>
<td>2d-strip-packing</td>
<td>0/2</td>
<td>0/2</td>
<td>0/2</td>
<td>0/2</td>
<td>0/2</td>
<td>1/2</td>
<td>1/2</td>
<td>1/2</td>
<td><strong>2/2</strong></td>
</tr>
<tr>
<td>crypto-sha</td>
<td>0/0</td>
<td>0/0</td>
<td>0/0</td>
<td>0/0</td>
<td>1/0</td>
<td>7/0</td>
<td>11/0</td>
<td><strong>13/0</strong></td>
<td>10/0</td>
</tr>
<tr>
<td>hardware-cec</td>
<td>0/22</td>
<td>0/23</td>
<td><strong>0/24</strong></td>
<td>0/22</td>
<td>0/22</td>
<td>0/23</td>
<td>0/22</td>
<td>0/21</td>
<td>0/21</td>
</tr>
<tr>
<td>hardware-manolios</td>
<td>0/4</td>
<td>0/5</td>
<td>0/5</td>
<td>0/5</td>
<td><strong>0/6</strong></td>
<td>0/6</td>
<td>0/6</td>
<td>0/6</td>
<td><strong>0/6</strong></td>
</tr>
<tr>
<td>hardware-velev</td>
<td>5/9</td>
<td>6/10</td>
<td>8/11</td>
<td>8/12</td>
<td>8/12</td>
<td><strong>8/13</strong></td>
<td>8/12</td>
<td>8/11</td>
<td>8/6</td>
</tr>
<tr>
<td>planning</td>
<td>6/3</td>
<td>6/5</td>
<td>7/4</td>
<td>7/4</td>
<td>8/4</td>
<td>9/3</td>
<td>9/4</td>
<td><strong>11/4</strong></td>
<td>10/4</td>
</tr>
<tr>
<td>scheduling</td>
<td>1/7</td>
<td>0/7</td>
<td>1/7</td>
<td>4/7</td>
<td>6/7</td>
<td>9/7</td>
<td>9/7</td>
<td><strong>11/7</strong></td>
<td><strong>12/7</strong></td>
</tr>
</tbody>
</table>

SAT / UNSAT

underlined best
static-008 versus static-256

- 2d-strip-packing
- argumentation
- bio
- crypto-aes
- crypto-des
- crypto-gos
- crypto-md5
- crypto-sha
- crypto-vpmc
- diagnosis
- fpga-routing
- hardware-bmc
- hardware-bmc-ibm
- hardware-cec
- hardware-manolios
- hardware-velev
- planning
- scheduling
- scheduling-pesp
- software-bit-verif
- software-bmc
- symbolic-simulation
- termination
bool restarting () {
    return conflicts >= limit &&
    average_RECENT_lbd ( ) > 1.25 * average_ALL_lbd ( );
}

void glucose_restart () { // same as static_uniform_restart
    restarts++;
    limit = conflicts + 50;
    backtrack (0);
}

- glucose level (LBD) of learned clause:
  - number of different decision levels in a learned clauses
  - calculated at the point the clause is learned during conflict analysis

- last 50 LBDs are stored and considered recent (explicit LBD queue)

- total average of all LBDs is simply \[ \frac{\text{sum}_\text{lbd}}{\text{conflicts}} \]

- for discussion of blocking restarts since Glucose 2.1 see the paper
Glucose uses *simple moving average* (SMA) for the average of recent LBDs and *cumulative moving average* (CMA) for the average of all LBDs and

\[
\text{simple} \quad \text{SMA}(n,w) = \frac{1}{w} \cdot (t_n + t_{n-1} + \ldots + t_{n-w+1}) \quad \text{with } n \geq w \geq 1
\]

\[
\text{cumulative} \quad \text{CMA}(n) = \text{SMA}(n,n)
\]

\[
\text{CMA}(n) = \text{CMA}(n-1) + \frac{t_n - \text{CMA}(n-1)}{n}
\]

\[
\text{SMA}(n,w) = \text{SMA}(n-1,w) + \frac{t_n}{w} - \frac{t_{n-w}}{w}
\]

requires \( \text{SMA}(n,50) > 1.25 \cdot \text{CMA}(n) \) to restart

and 50 conflicts have passed
we suggest to use *EMAs* instead of the “fast” *SMA* and/or “slow” *CMA*

exponential  \[ EMA(n, \alpha) = \alpha \cdot t_n + (1 - \alpha) \cdot EMA(n-1, \alpha) \]  with 0 < \alpha < 1  \[ a \approx \frac{2}{1+w} \]

alternative  \[ EMA(n, \alpha) = EMA(n-1, \alpha) + \alpha \cdot (t_n - EMA(n-1, \alpha)) \]

to restart version *average* requires  \[ EMA(n, 2^{-5}) > 1.25 \cdot CMA(n) \]

to restart version *ema-14* requires  \[ EMA(n, 2^{-5}) > 1.25 \cdot EMA(n, 2^{-14}) \]

and again in both cases that a certain number of conflicts say 50 have passed
LBD — fast $EMA$ of LBD with $\alpha = 2^{-5}$

restart — slow $EMA$ of LBD with $\alpha = 2^{-14}$ (ema-14)

inprocessing — $CMA$ of LBD (average)
### Comparing EMAs with SMA and CMA

#### Evaluating CDCL Restart Schemes

<table>
<thead>
<tr>
<th>solver</th>
<th>Glucose 4.0</th>
<th></th>
<th></th>
<th>Lingeling ba2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ss</td>
<td>es</td>
<td>ee</td>
<td>avg</td>
</tr>
<tr>
<td>restarts</td>
<td></td>
<td></td>
<td></td>
<td>avg</td>
</tr>
<tr>
<td>tot</td>
<td>163</td>
<td>163</td>
<td>165</td>
<td>178</td>
</tr>
<tr>
<td>sat</td>
<td>72</td>
<td>73</td>
<td>76</td>
<td>83</td>
</tr>
<tr>
<td>uns</td>
<td>91</td>
<td>90</td>
<td>89</td>
<td>95</td>
</tr>
<tr>
<td>avgc</td>
<td>192</td>
<td>166</td>
<td>167</td>
<td>145</td>
</tr>
</tbody>
</table>

Glucose 4.0 column ss correspond to the original Glucose version

column es to adding EMAs for only forcing restarts

column ee includes using EMA for blocking restarts too

column avg is Lingeling version average of Glucose version ee

columns eX correspond to Lingeling versions ema-X

using a slow EMA with $\alpha = 2^{-X}$ instead of CMA

---

Decision Heuristics and Restarts in SAT @ Deduktionstreffen 2016
double fast, slow;
...

bool analyze () {
    int lbd;
    ...
    slow += (lbd - slow)/(double)(1<<14);
    fast += (lbd - fast)/(double)(1<<5);
    ...
}

bool restarting () {
    return conflicts > limit && fast > 1.25 * slow;
}
data and source:  http://fmv.jku.at/evalrestart/evalrestart.7z

optimal restart interval varies with benchmark bucket
- for miters fast restarts essential
- for crypto benchmarks longer intervals necessary
- disabling restarts completely is bad
- Glucose restarts superior to Luby style

presented an EMA variant of the Glucose restart scheme
- simpler model, simpler to implement
- similar performance (slightly faster)

future work
- how to improve blocking of restarts
- restart intervals still not optimal: really need machine learning?
- combined SAT and Stock Market Analysis or SAT and Reinforcement Learning

originally proposed title for the POS’15 paper
solved SAT competition 2014 application instances (ordered by solving time)
Overall Conclusion

- decision heuristics
  - empirical insight: VMTF simpler and as good as VSIDS
  - simpler theoretical model
  - can we prove why these work?

- restarts
  - another look at restarts in SAT
  - simplified Glucose restart scheme, showed that it somehow works
  - clearly not where we want it to be (machine learning necessary?)